Complete Guide to Cross-Origin Resource Sharing (CORS)

keycdn.com/support/cors

In order to keep a website and its users secure from the security risks involved with sharing resources across multiple domains the use of CORS is recommended, but what is CORS? CORS, also known as Cross-Origin Resource Sharing, allows resources such as JavaScript and web fonts to be loaded from domains other than the origin parent domain.

These days, a web page commonly loads images, style sheets, scripts, etc. from other domains. Although, a few years ago due to security reasons, web fonts and AJAX (XML Http Requests) were normally restricted to the same-origin policy which restricted their use between domains. Now however, with the use of CORS, the browser and server can communicate to determine whether it is safe to allow a cross-origin request.

Why use CORS?

CORS was implemented due to the restrictions revolving around the same-origin policy. This policy limited certain resources to interact only with resources from the parent domain. This came with good reason as AJAX requests are able to perform advanced requests such as `POST`, `PUT`, `DELETE`, etc. which could put a website's security at risk. Therefore, the same-origin policy increased web security and helped prevent user abuse.

However, in some cases, it is quite beneficial to enable Cross-Origin Resource Sharing as it allows for additional freedom and functionality for websites. This is true in many cases these days for web fonts and icons which are often requested from another domain. In this case, with the use of HTTP headers, CORS enables the browser to manage cross-domain content by either **allowing or denying** it based on the configured security settings.

HTTP request headers

When a domain is requesting to interact with a resource on another domain, request headers are added from the first domain in order to use the Cross-Origin Resource Sharing feature. These are the HTTP request headers that may be associated with the requesting domain.

- `Origin`
- `Access-Control-Request-Method`
- `Access-Control-Request-Headers`

HTTP response headers

The domain who's resources are being requested can respond to the first domain with the following HTTP response headers based on what configuration options are set.

- `Access-Control-Allow-Origin`
- `Access-Control-Allow-Credentials`
- `Access-Control-Expose-Headers`
- `Access-Control-Max-Age`
- `Access-Control-Allow-Methods`
- `Access-Control-Allow-Headers`

Simple CORS example

Here is a simple CORS example of when a browser requests a resource from another domain. Let's say `domainx.com` makes a request to `domainy.com` for a particular resource. CORS uses HTTP headers to determine whether or not `domainx.com` should have access to that resource. The browser automatically sends a request header to `domainy.com` with

```
Origin: http://domainx.com
```

`domainy.com` receives that request and will respond back with either:

1. `Access-Control-Allow-Origin: http://domainx.com`
2. `Access-Control-Allow-Origin: *` (meaning all domains are allowed)
3. An error if the cross-origin requests are not allowed

Preflight CORS example

When certain, more complicated, types of requests are performed, the browser will insert additional preflight requests to validate whether they have the appropriate permissions to perform the action. A request is preflighted if any of the following conditions are met:

1. It uses an HTTP method other than `GET` or `POST`
2. Custom headers are set
3. The request body has a MIME type other than `text/plain`

Here is an example of a preflight request:

```
Origin: http://domainx.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-Custom-Header
```

If `domainy.com` is willing to accept the action, it may respond with the following headers:

```
Access-Control-Allow-Origin: http://domainx.com
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Headers: X-Custom-Header
```

A real-world example of how CORS works

CORS works by having the origin domain send HTTP request headers to the host domain that is hosting the resource. The example below shows that `https://www.keycdn.com` is the origin domain that is requesting a resource from the `Host: example.keycdn.com`.

Once `example.keycdn.com` receives the request, it responds by either allowing or denying the origin domain access to the requested resources based on the CORS settings configured. In the example below, it shows that the host responded with the response header of `Access-Control-Allow-Origin: *`. The `*` means all domains are allowed to access this resource.

CORS browser support

CORS is essentially supported by all modern browsers. If your browser doesn't support CORS it's likely a sign a sign that you should upgrade your browser version or change browsers. For example, Internet Explorer only offered partial support until IE 11. So if you're using a legacy version of IE consider upgrading or changing browsers entirely.

Spotting a CORS error

You'll likely notice that there is an issue related to CORS on your site if certain fonts or assets aren't loading properly. If you're using a CDN, this issue is likely to occur whenever if you decide to disable the CORS option in your Zone's settings. However, if you notice something is not rendering properly on your site you can use Google Chrome's DevTools to help troubleshoot the problem.

Once you open the Chrome DevTools, navigate to the Console panel. You might see other warnings and errors there but what you're looking for is something similar to the one below:

The domains have been blurred out for privacy reasons, however, this is basically saying that the font located at (e.g. `cdn.yourdomain.com/fonts/font.woff`) is blocking the origin (e.g. `yourdomain.com`) due to CORS. Therefore since the origin isn't allowed access, the font file cannot be pulled from the origin to the CDN.

jQuery with CORS

Due to their ability to perform advanced requests, cross domain AJAX requests are forbidden by default. With the use of CORS however, you have the ability to better define what methods are permitted. This helps increase website security all while having the ability to use features that otherwise would not be accessible.

Font Awesome CDN

Having CORS enabled is required to properly display Font Awesome icons when a CDN is implemented. If CORS is not enabled, the Font Awesome icons will not work and will look similar to the image below.

Thankfully, enabling CORS is easy and can be done directly from the KeyCDN dashboard or from the origin server. Font awesome files can be downloaded and delivered from your origin server, however, it is much more efficient to deliver them from a CDN URL.

Enabling CORS from the KeyCDN dashboard

In KeyCDN when a new Zone is added the **CORS** feature is automatically set to `enabled`. This setting can be set to `disabled` instead if you don't want this response header added.

Enabling CORS on the origin server

In case you want to handle the CORS settings on your side, you have the option of configuring CORS on your origin server. To enable CORS for static resources such as CSS, fonts, JavaScript, and images add the following code to your server.

Apache

```
<IfModule mod_headers.c>
    <FilesMatch "\.(ttf|ttc|otf|eot|woff|font.css|css|js|gif|png|jpe?g|svg|svgz|ico|webp)$">
        Header set Access-Control-Allow-Origin "*"
    </FilesMatch>
</IfModule>
```

Nginx

```
location ~ \.(ttf|ttc|otf|eot|woff|font.css|css|js|gif|png|jpe?g|svg|svgz|ico|webp)$ {
    add_header Access-Control-Allow-Origin "*";
}
```

Make sure you test your website with various browsers, especially FF and IE, as they are known to cause problems if CORS is not handled correctly. Also, if once you have enabled CORS and there is still an issue that persists, try purging your CDN cache.

CORS security

The same-origin policy does not allow for websites to communicate with each other's resources which can greatly limit a site's functionality. For this reason, CORS is a great feature for **minimizing the number of security risks** involved with web script sharing, while being able to utilize resources outside of the origin domain.

Having the ability to select which domains are allowed to access certain resources also gives added granularity to the resource sharing capability. When configured properly, CORS can easily integrate with other web services while keeping your website and users secure.